FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

DEI UC

Licenciatura em Engenharia Informática
Sistemas Distribuídos 2018/2019

infomusic

Arquivo de música com partilha de ficheiros
Meta 1 — 28 de outubro de 2018

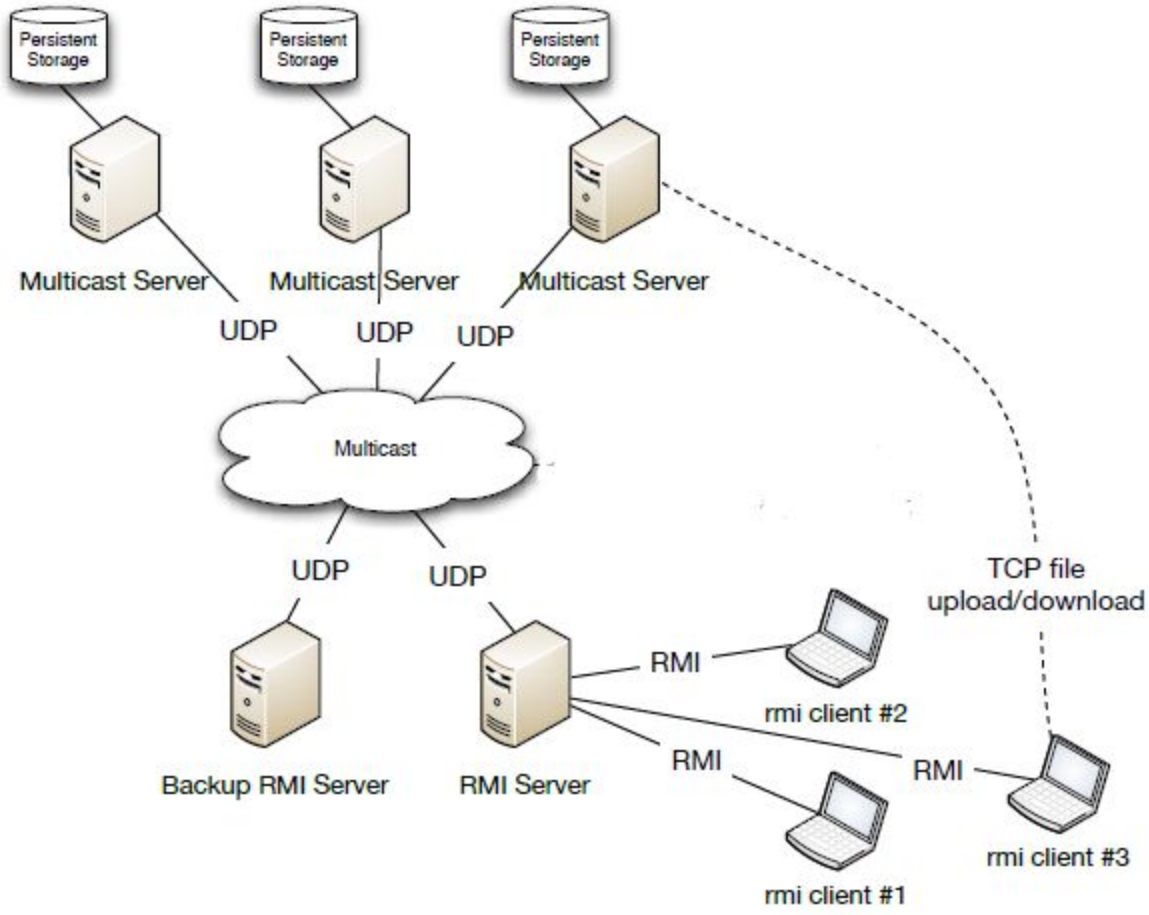
Hugo Miguel Brink Carvalho nº 2016227204
José Maria Campos Donato nº 2016225043

Introdução

Este projeto, realizado no âmbito da cadeira de Sistemas Distribuídos do 1º Semestre do 3º ano da Licenciatura em Engenharia Informática na Universidade de Coimbra consistiu em implementar um sistema de gestão de partilha de música. Neste sistema os utilizadores podem transferir ficheiros musicais que podem ser partilhados com outros utilizadores se assim o desejarem. Para a implementação deste projeto foi seguido uma arquitetura cliente-servidor que aplicou sockets TCP, UDP e Multicast. Foi usado Java RMI para criar uma camada de persistência de dados e foi garantida a disponibilidade da aplicação no caso de fail-over.

Arquitetura de Software

A figura abaixo, retirada do enunciado, mostra a arquitetura do sistema.



Servidor RMI

- Existem dois servidores RMI, um principal e um secundário. O secundário apenas substitui o principal no caso deste se avariar, tornando-se o servidor principal.
- Os servidores RMI disponibilizam um conjunto de métodos que podem ser invocados pelos clientes RMI. Quando um dos seus métodos é chamado, o RMI Server envia o pedido através de um datagrama UDP através da rede multicast para todos os servidores Multicast.

Servidor Multicast

- Os servidores de Multicast contêm toda a informação armazenada em bases de dados (postgresql).
- Eles recebem os pedidos através de datagramas multicast exceto se o cliente quiser realizar uma transferência de ficheiro de música. Neste específico caso é criada uma ligação TCP entre apenas 1 servidor Multicast e o cliente RMI que quer realizar esta operação, no entanto, no caso do download de uma música por parte do cliente, o servidor RMI comunica com o servidor Multicast para lhe dizer a música que o cliente.

Cliente RMI

- É através do cliente RMI que os utilizadores acedem às funcionalidades da infomusic.
- No cliente apenas se invocam os métodos remotos do servidor RMI e se mostram os resultados dos métodos ao utilizador.
- Apenas no upload e no download de ficheiros musicais é que o cliente RMI comunica diretamente ao Servidor Multicast através de uma ligação TCP.

Detalhes do funcionamento do servidor UDP

No protocolo que implementámos a estrutura principal utilizada é um hashmap (exemplos de hashmaps criados em baixo) e são enviados entre o servidor RMI e os servidores Multicast.

```
HashMap<String, String> hmap = new HashMap<>()
hmap.put("type", "changeData");
hmap.put("tableName", tableName);
hmap.put("columnType", columnType);
hmap.put("tableID", tableID+"");
hmap.put("newName", newName);
ConnectionFunctions.sendUdpPacket(hmap);
```

```
HashMap<String, String> hmap = new HashMap<>();
hmap.put("type", "addMusic");
hmap.put("name", name);
hmap.put("description", description);
hmap.put("duration", duration+"");
hmap.put("albumID", albumID+"");
hmap.put("artistID", artistID+"");
ConnectionFunctions.sendUdpPacket(hmap);
```

É indicado na chave **type** a operação para que ao chegar, por exemplo, ao Multicast Server seja tratado adequadamente (é alocada uma thread por cada pedido recebido no multicast, cada thread vai chamar a classe Threads que recebe o hashmap como parâmetro de entrada e como tem o parâmetro type vai tratar cada pedido de maneira diferente, exemplo à direita).

As restantes chaves são preenchidas dependendo de cada operação, como se pode ver nas duas imagens iniciais.

```
switch(this.map.get("type")) {
    case "login":
        try {
            treatLogin(this.map);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        break;
    case "register":
        try {
            treatRegister(this.map);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        break;
    case "verifyAdmin":
        try {
            treatVerifyAdmin(this.map);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        break;
    case "grantAdmin":
        try {
            treatGrantAdmin(this.map);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        break;
```

Para enviar a mensagem por UDP, o Hashmap é convertido numa string e, de seguida, em bytes através do método getBytes.

```
public static int sendUdpPacket(HashMap<String, String> map) {
    try {
        socket = new MulticastSocket();  // create socket without binding it (only for se
        byte[] buffer = map.toString().getBytes();
        InetAddress group = InetAddress.getByName(MULTICAST_ADDRESS);
        DatagramPacket packet = new DatagramPacket(buffer, buffer.length, group, PORT);
        socket.send(packet);
        System.out.println("d: sent udp with message: "+map.toString());
    } catch (IOException e) {
        e.printStackTrace();
        return 0;
    } finally {
        socket.close();
        return 1;
    }
}
```

Para receber a mensagem utilizamos o método “string2HashMap” que converte a String recebida (depois de usado o toString()) num Hashmap de novo.

```
String message = ConnectionFunctions.receiveUdpPacket();
HashMap<String, String> map = ConnectionFunctions.string2HashMap(message);
```

```
public static HashMap<String, String> string2HashMap(String string) {
    string = string.substring(1, string.length()-1);           //remove curly brackets
    String[] keyValuePairs = string.split( regex: ",");              //split the string to creat key-value pairs
    HashMap<String,String> map = new HashMap<~>();

    for(String pair : keyValuePairs)                        //iterate over the pairs
    {
        String[] entry = pair.split( regex: "=");                   //split the pairs to get key and value
        map.put(entry[0].trim(), entry[1].trim());          //add them to the hashmap and trim whitespaces
    }
    return map;
}
```

Detalhes do funcionamento do servidor RMI

Servidor:

No servidor existem diferentes métodos remotos que podem ser chamados pelos clientes RMI.

```
public interface InterfaceServer extends Remote {
    public boolean loginOrRegister(String username, String password, boolean isRegister) throws RemoteException;
    public boolean checkIfUserIsAdmin(String username) throws RemoteException;
    public boolean grantAdminToUser(String username) throws RemoteException;
    public void logout(String username) throws RemoteException;


    public String getTCPAddress() throws RemoteException;

    public boolean addMusic(String name, String description, Integer duration, Integer albumID, Integer artistID) throws RemoteException;
    public boolean addAlbum(String name, String genre, String description, String date, Integer artistID) throws RemoteException;
    public boolean addArtist(String name, String description) throws RemoteException;

    public boolean changeData(String tableName, String columnType, Integer tableID, String newName) throws RemoteException;
    public boolean writeAlbumReview(int albumToReviewID, int albumRating, String albumReview) throws RemoteException;

    public String searchDetailAboutAlbum(int albumToSearch) throws RemoteException;
    public String searchDetailAboutArtist(int artistToSearch) throws RemoteException;

    public String getTable(String table, String username) throws RemoteException;

    public boolean setMusicIDToDownload(String username, int musicID) throws RemoteException;
    public boolean shareMusicInCloud(String username, int musicIDToShare) throws RemoteException;

    public void subscribe(InterfaceClient c, String username) throws RemoteException;
    public boolean notifyUsersAboutAlbumDescriptionEdit(String username, int albumID) throws RemoteException;
    public boolean notifyUserAboutAdminGranted(String username) throws RemoteException;
    public String checkNotifications(String username) throws RemoteException;
    public boolean clearNotifications(String username) throws RemoteException;
    public boolean userEditAlbum(String username, int albumID) throws RemoteException;
    public String printOnlineUsers() throws RemoteException;


    public boolean uploadFileToTable(String table, String column, String fileLocation, Integer id) throws RemoteException;

}
```

- **loginOrRegister**
  - Método para a registar ou fazer login de um utlizador no programa
- **checkIfUserIsAdmin**
  - Método para verificar se um determinado utilizador é admin ou não
- **grantAdminToUser**
  - Método para fornecer permissoes de admin a um determinado utilizador
- **logout**
  - Método para fazer logout do utilizador no programa
- **getTCPAddress**
  - Método para arranjar um endereço IP para o rmi cliente fazer o download ou upload para os servidores multicast
- **addMusic, addAlbum, addArtist**
  - Métodos utilizados para adicionar músicas, álbuns ou artistas à base de dados do programa
- **changeData**
  - Método para alterar o nome ou descrição de um álbum, música ou artista
- **writeAlbumReview**
  - Método para escrever uma crítica e atribuir uma pontuação a um álbum

- **searchDetailAboutAlbum, searchDetailAboutArtist**
    - Métodos para pesquisar informação sobre um álbum ou um artista
- **getTable**
    - Método para ir buscar à base de dados os dados de uma tabela
- **setMusicIDToDownload**
    - Método onde o servidor rmi diz ao servidor Multicast qual a música que o utilizador pretende fazer o download
- **shareMusicInCloud**
    - Método para partilhar uma música com outro utilizador
- **subscribe**
    - Método para adicionar um user que esteja online ao array OnlineUsers
- **notifyUsersAboutAlbumDescriptionEdit, notifyUsersAdminGranted**
    - Método para adicionar à base de dados notificações para um utilizador no caso de um album ter sido editado ou de ter ganho permissoes de admin
- **checkNotifications**
    - Método para pesquisar todas as notificações de um utilizador enquanto ele esteve offline
- **clearNotifications**
    - Método para limpar as notificações de um utilizador depois de apresentá-los quando o utilizador faz login
- **userEditAlbum**
    - Método para adicionar à base de dados um utilizador alterou um certo album
- **printOnlineUsers**
    - Método que indica todos os utilizadores que estão online no programa

Cliente:

```
public interface InterfaceClient extends Remote {
    public void notifyAdminGranted() throws RemoteException;
    public void notifyAlbumChanges() throws RemoteException;
}
```

- **notifyAdminGranted, notifyAlbumChanges**
    - Método para enviar notificações a um certo utilizador online no caso de ter ganho permissões admin ou se um album que ele tenha editado tenha sido novamente editado

Distribuição de Trabalho

A distribuição do trabalho não foi dividida como sugerido pelo professor entre a parte RMI e a parte Multicast mas sim em pequenas subdivisões entre as funcionalidades pedidas. Numa visão global, o aluno **José** ficou encarregue da ligação do Multicast Server à base de dados, pela parte do Multicast Server e por alguns dos métodos RMI Server e pelo Javadoc.
O aluno **Hugo** ficou encarregue pelos menus do RMI Client, alguns dos métodos do RMI Server, pelo FailOver. O relatório foi realizado por ambos os alunos.
Embora esta divisão seja necessária houve sempre uma entreajuda, sabendo cada elemento em que pé se encontrava o projeto e havendo também diversas “working sessions” onde programamos juntos a mesma parte do projeto.

Testes de Software

1. RMI Backup
    a. Ligar 2 servidores RMII;
    b. Verificar se um se torna principal e o outro secundário
    c. Desligar o RMI principal.
    d. Verificar se outro RMI se liga e torna principal;
    e. Ligar novamente o RMI Server;
    f. Verificar se este se torna secundário.

2. RMI Client
    a. Correr servidor RMI;
    b. Correr cliente RMI;
    c. Fechar servidor RMI;
    d. Verificar se o cliente fica à espera que o RMI Server retome

3. Manter sessão
    a. Correr 2 servidores RMI;
    b. Correr cliente RMI;
    c. Fechar servidor RMI;
    d. Verificar se o RMI Cliente se liga ao RMI Server Backup
    e. Verificar se, ao ligar, mantêm a sessão

4. Login
    a. Inserir dados errados
    b. Verificar se é aceite pelo sistema

5. Notificação
    a. Ligar 1 servidor RMI e 2 cliente RMI (um editor e outro não) e 1 servidor Multicast
    b. Tornar o que não é editor em editor
    c. Verificar se recebe a notificação

6. Notificação offline
    a. Ligar 1 servidor RMI e 1 cliente RMI (editor) e 1 Multicast
    b. Tornar o um cliente que não é editor em editor
    c. Ligar um o cliente que agora é editor
    d. Verificar se esse recebe a notificação que agora é editor

7. Upload
    a. Ligar 1 servidor RMI e 1 cliente RMI e 1 Multicast

    b. Fazer o upload de uma música

8. Download
    a. Ligar 1 servidor RMI e 1 cliente RMI e 1 Multicast
    b. Fazer download de uma música

9. Partilha de músicas
    a. Ligar 1 servidor RMI e 2 cliente RMI e 1 Multicast
    b. Escolher uma música para partilhar com o segundo cliente
    c. Tentar fazer o download da música alterada com o cliente com a qual foi partilhada

10. Sistema em 2 máquinas
    a. Ligar 1 servidor Multicast e 1 cliente RMI
    b. Ligar 1 servidor multicast, 2 servidores RMI e 1 cliente RMI
    c. Fazer pedidos com os clientes RMI

Resultados

Todos os testes, à exceção do 10, mostrados em cima foram bem sucedidos!

10. Primeiro não funcionou porque virtualbox entra em conflito. Após a desinstalação do programa virtualbox, funcionou corretamente.